공통 문서 구조 추출을 통한 XML DTD의 관계형 데이터 베이스 스키마 변환 기법

안 성 은† · 최 황 규††

요 약

XML은 W3C에 제안된 마크업 언어로 HTML의 단순함과 SGML의 복잡함을 극복하여, 웹 상에서 데이터를 표현하고 교환하기 위한 표준으로 등장하고 있다. XML 문서를 질의 처리하기 위한 방법으로 XML 문서 전용 질의 언어가 개발되고 있지만, 데이터의 양이 증가한다면 결국 막대한 양의 데이터를 처리 할 데이터베이스 시스템을 필요하게 된다. 본 논문에서는 XML DTD를 관계형 데이터베이스 시스템 스키마로 변환하는 기법을 제안한다. 제안된 기법은 XML 데이터의 스키마 역할을 하는 DTD의 트리 구조를 생성하여, XML 데이터들의 공통구조와 비공통구조를 추출한 후 관계형 데이터베이스 스키마를 추출하는 기법이다. 추출된 관계형 데이터베이스 스키마는 기존의 방법들에 비해 생성 테이블 수가 적으며, 널(NULL)값의 출현을 감소시킨다. 또한, 제안기법은 XML 데이터를 보다 적은 테이블로 맵핑(mapping)시킴으로써 데이터 검색 시 참조 테이블 수를 감소시킬 수 있으며 질의 처리 시에도 성능 면에서 우수함을 보인다.

A Transformation Technique of XML DTD to Relational Database Schema Based On Extracting Common Structure in XML Documents

Sung-Eun Ahn† · Hwang-Kyu Choi††

ABSTRACT

XML is emerging as a standard data format to exchange and to present data on the Web. There are increasing needs to efficiently store and to query XML data. In this paper, we propose a new schema transformation algorithm based on a common structure extracting technique from XML documents. The common structure is shared by all XML documents referenced by DTD and the uncommon structure is ununiformly appeared on all XML documents referenced by DTD. Based on the extracted common and uncommon structures, we transform XML DTD into relational database schema. We conduct a performance evaluation based on the number of the generated tables, the size of the record, query processing time and the number of joins on the query. The performance of our algorithm is compared with the existing algorithms, then in most cases, our algorithm is better than the existing ones with respect to the number of the generated tables and appearance of NULL values in the tables.

키워드 : XML(Extensible Markup Language), DTD(Document Type Definition), 관계형 데이터베이스(Relational Database), 스키마(Schema), 카디널리티 연산자(Cardinality Operator)

1. 서 론

XML[16]은 최근 웹 상에서 데이터 교환의 표준 형식으로 주목을 받고 있으며 사용자가 문서의 구조를 정의 할 수 있기 때문에 다양한 형태의 데이터가 XML 문서로 표현 될 수 있다. 이러한 점은 XML이 이 기종 컴퓨터 사이에서 데이터 교환의 매체로 사용 될 수 있다는 것을 암시한다. XML은 DTD(Document Type Definition)[16]를 통하여 문서 자체에 문서의 구조를 기술하고 있다. 즉, 문서의 구조를 사용자가 원하는 대로 정의 할 수 있으며, 이러한 구조적 유동성은 다양한 형태의 데이터를 XML로 표현할 수 있게 해준다. 웹에서 운용되는 데이터가 동일한 형태로 저장, 처리 될 수 있음을 의미한다.

이러한 다양한 형태의 XML 문서를 효율적으로 저장·관리하고, 이들 문서로부터 유용한 정보를 추출하기 위해서는 데이터베이스 시스템의 활용이 필수적이며, 현재 이와 관련된 연구분야 중 XML 데이터 형식을 데이터베이스 스키마로 변환하고자 하는 연구가 활발히 진행되고 있다. 이러한 변환 방법에는 XML 데이터 형식을 기존의 관계형 데이터베이스 시스템이나 객체지향 데이터베이스 시스템의 데이터 형식으로 변환하기 위한 연구와[1, 2, 4], XML 전용 데이터베이스 시스템의 개발에 대한 연구들이[3, 5-7] 진행되고 있다. 이들 연구 결과들은 각각의 특징과 장단점을 가지고 있으며, 여러

※ 본 논문은 강원대학교 BK21 사업단 지원에 의한 연구 결과의 일부임.
† 준 회 원 : 강원대학교 대학원 컴퓨터정보통신공학과
†† 정 회 원 : 강원대학교 전기전자정보통신공학부 교수
논문접수 : 2002년 9월 30일, 심사완료 : 2002년 11월 6일

데이터베이스 시스템에서도 이러한 방법들을 수용하여 선택적으로 사용할 수 있도록 하는 기능을 제공하고 있다.

본 논문에서는 XML 데이터를 관계형 데이터 베이스 시스템의 데이터 형식으로 변환하기 위한 새로운 알고리즘을 제안한다. 제안된 기법은 XML 데이터의 스키마 역할을 하는 DTD의 트리 구조를 생성하여, XML 데이터들의 공통구조와 비 공통구조를 추출한 후 관계형 데이터베이스 스키마를 추출하는 기법이다. 추출된 공통구조는 동일한 DTD를 사용하는 모든 XML 문서에 반드시 존재하는 노드들이고, 비 공통구조는 각각의 XML 문서에 비 규칙적으로 출현하는 노드들이다. 이러한 공통구조와 비 공통구조에 대하여 관계형 데이터베이스 스키마가 각각 따로 추출된다.

본 논문에서는 제안된 기법에 대한 성능 평가를 위해 XMLDBMS Java Package[14]의 소스 코드를 수정하여 공통구조와 비 공통구조를 추출하는 시스템을 구현하였다. 추출된 관계형 데이터베이스 스키마는 기존의 방법들[1, 14]에 비해 생성 테이블 수가 적으며, 널(NULL)값의 출현을 감소시킨다. 또한, 제안기법은 XML 데이터를 보다 적은 테이블로 맵핑(mapping)시킴으로써 데이터 검색 시 참조 테이블 수를 감소시킬 수 있어 질의 처리 성능 면에서 우수함을 보인다.

본 논문은 먼저 2장에서 XML 데이터를 기존의 데이터베이스 시스템의 데이터 형태로 변환하기 위한 관련 연구들을 살펴보고, 3장에서는 XML 문서의 공통구조 추출을 통한 관계형 데이터베이스 스키마 추출 기법을 설명한다. 또한, 제안 기법의 성능 평가를 위하여 XML 문서의 DTD를 관계형 데이터 베이스 스키마로 변환하는 시스템과 성능평가 결과를 4장에서 기술하고, 마지막으로 5장에서 결론을 맺는다.

2. 관련 연구

본 장에서는 XML 데이터를 기존의 데이터베이스 시스템의 데이터 형식으로 변환하기 위한 관련 연구들을 살펴본다. 관계형 데이터베이스 시스템에 XML 데이터를 저장하기 위해서는 XML 데이터를 관계형 데이터베이스로 맵핑하는 과정이 필요하다. 이러한 맵핑 과정은 XML 데이터 구조를 관계형 데이터베이스 스키마로 변환하는 작업이 필요하며, 본 장에서는 기존의 관계형 데이터베이스 스키마 추출 알고리즘을 살펴본다.

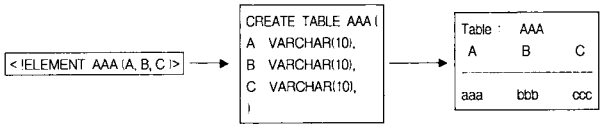
```
< person id = 1 age = 55 >
      < name > Peter </name >
      < address > 4711 Fruitdale Ave. < /Address>
            <person id = 3 age = 22 >
                  < name > John < /name >
                  < address > 5361 Columbia Ave. < /address >
                  < hobby > swimming < /hobby >
            < /person >
      < /child >
      <child>
            < person id = 4 age = 7>
                  < name> David < /name >
                  < address > 4711 Fruitdale Ave. < /address >
            < /person >
      < /child >
< /person >

< person id = 2 age = 38 child = 4 >
      < naem > Mary < /name >
      < address > 4711 Fruitdale Ave. < /address >
      < hobby > painting < /hobby >
< /person >
```

(그림 1) 예제 XML 문서

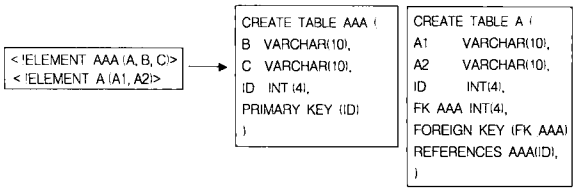
XML 문서를 관계형 데이터베이스 시스템에 저장하기 위해 스키마를 추출하는 방법에는 여러 가지가 있을 수 있다. [4]는 관계형 데이터베이스 스키마의 속성과 값을 기준으로 다양한 스키마 추출 알고리즘을 제안했다. 가장 간단하게는 (그림 1)의 XML 데이터의 모든 요소(element, attribute, text-value)들을 하나의 테이블에 모두 저장하는 방법이 있다. 두 번째 방법으로는 같은 이름을 가지는 요소들을 하나의 테이블에 함께 저장하는 방식이다. 즉, (그림 1)의 XML 문서에서 age, name, address, child, hobby와 같은 요소를 각각의 테이블에 저장하는 방법으로 많은 테이블을 필요로 한다. 세 번째 방법으로는 첫 번째 방법의 데이터 값들을 따로 분리해 정수형 데이터와 문자열 데이터의 테이블에 저장하는 방법이다.

[2]는 XML 문서를 Element, Attribute, Text의 세 가지 형태로 구분하여 모델링 하였다. 물론, 주석(comment) 등 기타 다른 노드 타입이 존재 할 수 있지만 이 방법에서는 그러한 것들은 고려하지 않았으며 XML 문서를 저장하는 데이터베이스 스키마는 DTD나 Element 타입과는 독립적이다. XML 문서를 저장하는 릴레이션은 Element, Attribute, Text, Path 네 가지로 구성된다. Element 테이블은 Element 노드들에 대한 정보를 저장하는데 테이블 속성은 문서식별자(docID), 경로식별자(pathID), 인덱스, 위치정보이다. Attribute 테이블은 Attribute 노드들에 대한 정보를 저장하는데 테이블 속성으로는 문서식별자(docID), 경로식별자(pathID), attribute 값, 위치정보이다. Text 테이블은 Text 노드에 대한 정보를 저장하며 테이블 속성으로는 문서식별자(docID), 경로식별자(pathID), Text 값, 위치정보를 가진다. 마지막으로 Path 테이블은 간단한 경로에 관한 정보를 저장하는데 테이블 속성은 경로표현(pathexp), 경로식별자(pathID)를 가진다.

XML 문서를 분해하여 각각의 노드들의 타입에 따라 저장을 하면 XML 문서를 저장하는 데이터베이스 스키마는 DTD와 노드 타입에 독립적이다. 따라서 어떠한 XML 문서도 네 개의 테이블을 기반으로 하여 관리 할 수 있지만, XML 데이터의 스키마 역할을 하는 DTD를 이용 할 수 없다.



(그림 2) XMLDBMS 예 1



(그림 3) XMLDBMS 예 2

[14]는 가장 일반적인 스키마 추출 방법의 하나이며, XML 데이터와 관계형 데이터베이스 시스템 사이에서 데이터 교환을 담당하는 미들웨어로써 JAVA Packages 형태로 배포 중이다. (그림 2)에서와 같이 DTD 문서에서 부모 노드는 하나의 테이블로 맵핑되며 자식노드들은 테이블 속성으로 맵핑된다. DTD의 구조가 복잡해지면 부모 테이블 노드와 자식 테이블 노드를 연결하기 위해 (그림 3)과 같이 외래키(foreign key)를 생성한다. XMLDBMS에서 XML 문서의 모든 부모 노드는 관계형 데이터베이스 시스템의 테이블로 맵핑된다. 이러한 방법은 하나의 XML 문서가 다수의 테이블로 분산된다는 단점과, '?', '*' 연산자로 인한 테이블 내의 널(NULL) 값의 출현이 빈번하다는 단점이 있다.

[1]은 XMLDBMS 방법의 단점인 다수의 테이블 생성을 극복한 방법으로써 XML 문서의 스키마 역할을 하는 DTD를 트리 구조로 모델링 하였다. 관계형 데이터베이스 스키마로 맵핑시 생성 테이블의 수를 줄이기 위해 'root' 노드와 '*', '+' 연산자를 가지는 엘리먼트 노드를 테이블로 맵핑시켰으며, 순환 관계를 가지는 복잡한 DTD를 구조를 위해 순환 노드를 테이블로 맵핑시킨다. DTD의 속성타입 중 다수의 데이터를 가지는 'IDREFS' 타입을 또한 테이블로 맵핑시킴으로써 속성문제를 해결했다. 하지만 '*', '?' 노드로 인한 테이블 내의 널(NULL) 값의 존재는 단점으로 남았다.

[3]는 유효하지 않은 XML 문서의 처리까지 고려한 방법으로써 XML 문서가 주어진 DTD와 일치하지 않을 경우, 데이터 마이닝(Data Mining) 알고리즘을 이용해 DTD를 추출한다. 또한, [11]은 복잡한 DTD를 간단하게 만드는 알고리즘을 제안했다.

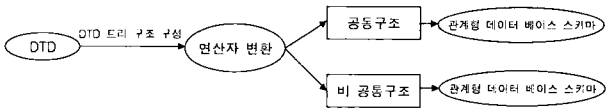
본 논문에서는 이러한 기존의 스키마 추출 알고리즘의 단점들을 극복하기 위한 방법으로써 XML 문서의 공통구조를 추출하여 관계형 데이터베이스 시스템에 XML 데이터를 저장하기 위한 스키마 추출 기법을 제안한다.

3. XML DTD의 관계형 데이터베이스 스키마 변환 기법

XML 문서는 XML 데이터들의 스키마 역할을 하는 DTD (Document Type Definition)[16]를 가지고 있다. DTD는 XML 문서의 구조를 정의한 것으로써, DTD가 강제적인 요소는 아니지만 도서목록이나 은행의 고객관리 정보와 같은 데이터를 XML 문서로 작성 할 때에는 문서의 구조에 관한 정보를 가지고 있는 DTD가 반드시 필요할 것이다. 앞으로 웹 상에서 다루어지는 데이터가 XML 형태로 표현된다면 특정분야의 데이터를 위한 일반화된 DTD가 필요할 것이며, 각각의 수많은 XML 문서들은 이러한 일반화된 DTD를 기반으로 작성 될 것이다.

본 장에서는 XML 문서의 구조 정보를 담고 있는 DTD를 이용하여 동일한 DTD를 사용하는 XML 문서들의 공통 구조를 추출하는 기법을 제안하며, 제안기법을 이용하여 XML 데이터를 관계형 데이터베이스 시스템에 저장하기 위한 스키마를 추출하는 기법을 설명한다.

3.1 기본 알고리즘



(그림 4) 알고리즘 구성도

본 논문에서는 (그림 4)에서와 같이 DTD를 입력받아 관계형 데이터 베이스 스키마를 추출하는 기법을 제안한다. 제안된 기법은 XML 데이터의 스키마 역할을 하는 DTD의 트리 구조를 생성하여, XML 데이터들의 공통구조와 비 공통구조를 추출한 후 관계형 데이터베이스 스키마를 추출하는 방법이다. 추출된 공통구조는 동일한 DTD를 사용하는 모든 XML 문서에 반드시 존재하는 노드들이고, 비 공통구조는 각각의 XML 문서에 비 규칙적으로 출현하는 노드들이다. 이러한 공통구조와 비 공통구조에 대하여 관계형 데이터베이스 스키마가 각각 따로 추출된다.

XML 문서의 DTD(Document Type Definition)의 요소(element) 선언부에는 각 노드의 발생 빈도를 결정해주는 카디널리티 연산자(Cardinality Operator)를 가지고 있다[16]. 카디널리티 연산자는 노드 자신에게만 적용되기 때문에 부모와 자식 노드로 연결되는 연산자를 적용하기 위해 연산자 변환을 적용한다. 카디널리티 연산자를 이용해 부모 자식 노드간의 발생 빈도를 설정하여 DTD 그래프에 적용함으로써 XML 문서들의 공통구조와 비 공통구조를 추출하며, 추출된 공통구조와 비 공통구조에 대한 관계형 데이터베이스 스키마를 각각 따로 생성한다.

3.2 카디널리티 연산자 변환

본 절에서는 제안기법의 핵심인 연산자 변환을 설명한다. <표 1>은 XML 문서의 DTD의 요소 선언부에서 각 노드의 발생 빈도를 설정해 주는 카디널리티 연산자이다. [none]은 노드가 단 한 개만 존재하고, '?'는 있거나 없거나 둘 중 한 가지 경우이다. '*'는 없거나 여러 개 존재하고, '+'는 한 개 이상 여러개 존재한다는 의미이다. 카디널리티 연산자는 자

기 자신에게만 적용 가능한 연산자이다. 카디널리티 연산자를 부모노드에서 자식노드까지 적용시키기 위해서 <표 2>와 같은 연산자 변환 설정을 정의한다. 즉, <표 2>와 같이 부모와 자식노드의 '연산자 변환'을 표현 할 수 있다. 부모노드와 자식노드의 연산자 변환을 정의한 것으로 괄호 안의 숫자들은 노드의 발생 범위를 나타낸다. 예를 들어 다음과 같은 DTD가 있다.

<!ELEMENT AAA (A, B+)>
<!ELEMENT B (B1?, B2*)>

여기서 "B/B1"에 대한 연산자 변환은 +? → *이다. 즉, "B/B1"은 (0, 1, 2, 3, …)의 노드 개수 범위를 가진다. 다시 말하면 "B"라는 노드는 한번이상 여러 번 문서에 발생할 수 있지만, "B/B1"이라는 노드는 문서에 발생하지 않을 수도 있다. 위와 같은 방법으로 루트노드에서 마지막 자식노드까지 연산자 변환을 적용하면, 모든 노드는 'n, *, +'로 나타낼 수 있다. 연산자 변환 후 '+' 'n'연산자를 가지는 노드는 동일한 DTD를 바탕으로 작성된 어떠한 XML 문서에도 반드시 존재하는 노드이다. 본 논문에서는 연산자 변환 기법을 XML 문서의 DTD 트리 구조에 적용함으로써 공통구조를 추출한다.

〈표 1〉 카디널리티 연산자

카디널리티 연산자	설 명
[none]	오직 한 개의 자식 요소만 허용 - 필수적인 요소
?	자식 노드가 없거나 한 개 - 선택적인 단일 요소
*	자식 요소가 없거나 다수 - 선택적 요소
+	자식 노드가 하나이거나 다수 - 필수적인 요소

〈표 2〉 카디널리티 연산자 변환

[none]을 n으로 표기	
nn → n (1)	** → * (0, 1, 2, 3, …)
n? → ? (0, 1)	*n → * (0, 1, 2, 3, …)
n* → * (0, 1, 2, 3, …)	*? → * (0, 1, 2, 3, …)
n+ → + (1, 2, 3, 4, …)	*+ → * (0, 1, 2, 3, …)
?? → ? (0, 1)	++ → + (1, 2, 3, 4, …)
?n → ? (0, 1)	+n → + (1, 2, 3, 4, …)
?* → * (0, 1, 2, 3, …)	+? → * (0, 1, 2, 3, …)
?+ → * (0, 1, 2, 3, …)	+* → * (0, 1, 2, 3, …)

3.3. 카디널리티 연산자의 속성 값 적용

본 절에서는 XML 문서의 카디널리티 연산자 변환 기법을 XML 속성요소에 적용하기 위한 방법을 기술한다.

XML 문서는 엘리먼트 노드만으로 구성되지 않는다. 즉, XML 문서는 속성요소를 포함하는데 카디널리티 연산자 변환을 속성요소에 적용하기 위한 방법은 다음과 같다. <표 3>에서와 같이 XML 문서의 DTD 속성 선언부에는 각각의 속성 값들의 필수 존재 여부를 결정해주는 속성 기본 파라미터 값들이 있다. '#REQUIRED'는 속성 값이 반드시 존재해야 한다는 의미이므로 카디널리티 연산자의 'n'으로 표현될 수 있으며, '#IMPLIED'와 '#FIXED'는 존재할 수도 있고 없을 수도 있다는 의미이므로 카디널리티 연산자의 '?'로 표현 될 수 있다. 즉, DTD 속성 요소의 기본 파라미터를 이용해 카디널리티 연산자를 적용함으로써 속성의 공통요소를 추출 할 수 있다.

〈표 3〉 속성 기본 파라미터 연산자 적용

속성 기본 파라미터	설 명	표기
#REQUIRD	속성은 반드시 요소의 모든 인스턴스 안에 있어야 한다	n
#IMPLIED	속성은 문서 안에 있을 수도 있고, 없을 수도 있다.	?
#FIXED	속성은 문서 안에 있을 수도 있고, 없을 수도 있다. 하지만 속성이 있다면 기본 값을 가진다.	?

DTD의 속성 타입 부분에서 주의 깊게 보아야 할 부분은 'ID'와 'IDREF' 'IDREFS'이다. 'ID'는 각각의 엘리먼트 노드들의 식별자 역할을 하는 요소로써 'IDREF'와 'IDREFS'를 속성으로 가지는 엘리먼트 노드들의 레퍼런스 역할을 한다. 'IDREF' 요소는 속성 값으로 'ID'값을 하나 가질 수 있고, 'IDREFS' 요소는 속성 값으로 여러 개의 'ID'값을 공백으로 구분하여 가질 수 있다. 그러므로 'IDREFS'는 카디널리티 연산자의 '+'으로 표현 될 수 있다. 즉, 'IDREFS'는 속성 값의 노드이지만, 엘리먼트 노드가 '+' 연산자를 가지고 있는 것과 같은 효과를 나타내기 때문에 'IDREFS' 요소를 관계형 데이터베이스 스키마 형태로 변환할 때는 엘리먼트가 '+' 연산자를 가지고 있는 것처럼 처리해야 한다.

3.4 공통구조 추출 기법

이 절에서는 앞 절에서 설명한 연산자 변환 기법을 이용하여 XML 문서의 DTD로부터 공통구조를 추출하는 기법을 설명한다. XML 문서의 DTD로부터 공통구조를 추출하는 과정은 다음과 같다.

(1) DTD를 입력받아 유효성 검증 및 트리구조 구성
(2) DTD 내의 카디널리티 연산자 정보를 추출
(3) DTD 트리 구조에 연산자 변환을 적용하여 공통구조와 비 공통구조 추출

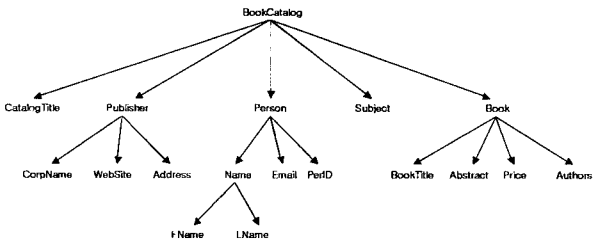
본 절에서는 DTD의 트리 구조를 구성하여 공통구조를 추출하는 과정을 그래프를 이용하여 설명한다.

```
<!ELEMENT BookCatalog (CatalogTitle, Publisher+, Person+,
  Subject+, Book+)>
<!ELEMENT CatalogTitle (#PCDATA)>
<!ELEMENT Publisher (CorpName, Website*, Address?)>
<!ELEMENT CorpName (#PCDATA)>
<!ELEMENT Website (#PCDATA)>
<!ELEMENT Address (#PCDATA)>
```

```
<!ELEMENT Person ( Name, Email* ) >
<!ATTLIST Person perID ID #REQUIRED >
<!ELEMENT Name ( FName?, LName ) >
<!ELEMENT Email ( #PCDATA ) >
<!ELEMENT FName ( #PCDATA ) >
<!ELEMENT LName ( #PCDATA ) >
<!ELEMENT Subject ( #PCDATA ) >
<!ELEMENT Book ( Abstract?, BookTitle, Price?) >
<!ATTLIST Book authors IDREFS #REQUIRED >
<!ELEMENT Abstract ( #PCDATA ) >
<!ELEMENT BookTitle ( #PCDATA ) >
<!ELEMENT Price ( #PCDATA ) >
```

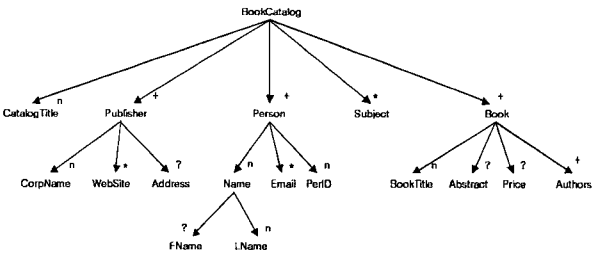
(그림 5) BookCatalog DTD

(그림 5)는 본 논문에서 예로 사용할 'BookCatalog.dtd'로 도서목록에 관한 데이터를 다루는 XML 문서를 위한 DTD이다. DTD를 방향성 그래프 형태로 모델링하여 그래프를 그리면 (그림 6)과 같다. (그림 6)에 연산자를 적용하면 (그림 7)과 같으며 각각의 카디널리티 연산자는 XML 문서의 노드 발생 빈도를 나타낸다.

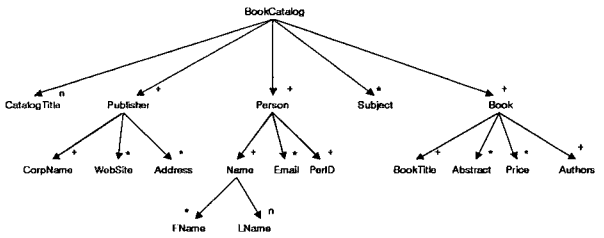


(그림 6) DTD 그래프

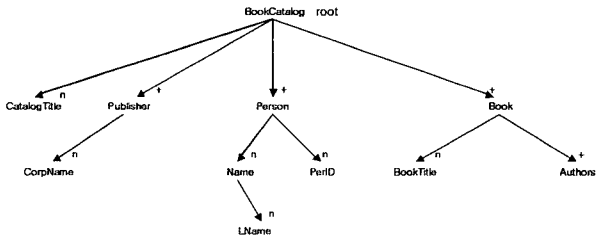
즉, (그림 7)에서 '+'로 표시된 노드는 문서 내에서 1번 이상 나타나고, '*'로 표시된 노드는 0번 이상 나타날 것이며, '?'로 표시된 노드는 문서에 있을 수도 있고 없을 수도 있다. 'n'으로 표시된 노드는 문서에 한번만 나타날 것이다. 또한, (그림 7)에 연산자 변환을 적용하면 (그림 8)과 같다. (그림 8)에서 '+', 'n'연산자를 가지는 노드들은 전체 XML 문서에 공통적으로 존재하는 노드들이며 "공통스키마"라 칭하겠다. 이러한 공통스키마만을 분리하여 그린 그래프는 (그림 9)와 같으며 "공통 스키마 그래프"라 칭한다. 공통 스키마 그래프는 동일한 DTD를 사용하는 전체 XML 문서들에 반드시 존재하는 노드들만으로 구성된 것으로, 공통 스키마를 바탕으로 관계 데이터베이스에 저장하기 위한 공통 릴레이션 스키마를 추출한다.



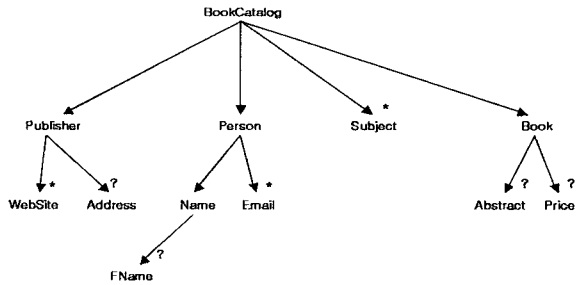
(그림 7) DTD 연산자 그래프



(그림 8) DTD 연산자 변환 그래프



(그림 9) 공통 스키마 그래프



(그림 10) 비 공통 스키마 그래프

초기 DTD의 카디널리티 연산자 구조는 [1]에서 사용된 기법을 통하여 간단하게 된 상태이며, ENTITY나 NOTATION과 같은 DTD 구성 요소는 고려하지 않았다. 이때 공통 릴레이션 스키마 추출 과정은 다음과 같다.

(1) DTD 그래프 생성 : DTD의 유효성을 검증하여 초기화된 DTD를 통해 각각의 노드정보를 가져온다.
(2) 연산자 변환 : DTD내의 카디널리티 연산자 정보를 추출하여 연산자를 변환한다.
(3) 공통 스키마 추출 : 카디널리티 정보를 통해 공통 스키마를 생성할 노드를 분리한다.
(4) 공통 릴레이션 스키마 생성 : 테이블 노드는 XML 문서의 노드들 중에서 관계형 데이터베이스의 테이블로 맵핑되는 노드를 말한다. 공통 스키마에서 'root'와 '+' 연산자를 가지는 노드는 하나의 테이블로 맵핑되며, 부모와 자식 테이블 노드와는 외래키(foreign key)로 참조된다. 테이블 노드의 자식노드들은 테이블의 속성 값이 된다. 또한, 속성 기본 파라미터 값이 'IDREFS'인 노드 또한 하나의 테이블로 맵핑되며 부모 노드와 외래키로 참조된다. 각각의 테이블 노드에 대한 기본키(primary key) 필드를 생성한다. ID 속성이 존재하면 사용하며, 없는 경우는 자동으로 생성한다. 공유되는 엘리먼트와 순

환하는 노드를 위해 parent_elm와 root_elm 필드를 생성한다. 공통 릴레이션 스키마는 전체 XML 문서들에 공통적으로 반드시 존재하는 노드들만으로 구성 된 것으로 널(NULL)값을 포함하지 않는다.

3.5 비 공통구조 추출 기법

(그림 10)의 비 공통 스키마 그래프는 DTD 그래프에서 공통 스키마를 제외한 나머지 노드들을 나타내는 것으로 데이터의 손실을 막기 위해서 공통 스키마와 분리하여 "비 공통 스키마"라 칭하며, 관계 데이터베이스 시스템에 저장하기 위한 비 공통 릴레이션 스키마를 따로 추출한다.

비 공통 스키마에 포함되는 데이터들은 공통 스키마의 테이블 노드들의 자식노드들로 구성된다. 즉, 비 공통 스키마에 속하는 데이터는 반드시 하나의 공통 스키마의 테이블 노드의 자식노드가 되기 때문에, 비 공통 릴레이션 스키마는 각각의 공통 릴레이션 스키마의 부모 테이블을 외래키로 참조하고, id, root_elm, parent_elm, element_name, other_values, foreign key 필드를 자동 생성한다. element_name 필드는 비 공통 노드의 엘리먼트 이름 값을 가지며, other_values 필드는 실제 데이터가 입력된다. foreign key 필드는 공통 릴레이션 스키마 테이블 노드들의 '이름'을 "FK_이름" 형태로 자동 생성한다.

3.6 스키마 추출 결과

스키마 추출 결과는 SQL Script이며, (그림 11)은 BookCatalog.dtd를 입력받은 후의 결과 SQL Script이다. (그림 9)의 공통 스키마에서 테이블 노드는 BookCatalog(root), Publisher(+), Person(+), Book(+), Authors(IDREFS)이며 (그림 11)의 결과 SQL문에서처럼 각각의 테이블 노드는 테이블을 생성하며 id, root_elm, 'FK_부모테이블이름' 필드를 생성한다. 또한 테이블 노드의 자식노드들은 테이블의 속성으로 맵핑되는 것을 확인 할 수 있다.

비 공통 스키마의 경우는 앞 절에서 설명한 것처럼 공통 릴레이션 스키마의 부모 테이블을 외래키로 자동 생성하는데 (그림 11)의 OTHER_NODES 테이블에서 FK_BOOKCATALOG, FK_PUBLISHER, FK_PERSON, FK_BOOK, FK_AUTHORS 필드가 생성 된 것을 확인할 수 있다.

```
CREATE TABLE BOOKCATALOG (
        BOOKCATALOG_CATALOGTITLE        VARCHAR (500),
        ID                              INT,
        ROOT_ELM                        VARCHAR (500)
        PRIMARY KEY (ID)
);
CREATE TABLE PUBLISHER (
        FK_BOOKCATALOG                  INT,
        PUBLISHER_CORPNAME              VARCHAR (500),
        ROOT_ELM                VARCHAR (500),
        ID                      INT,
        FOREIGN KEY (FK_BOOKCATALOG) REFERENCES BOOKCATILOG (ID),
        PRIMARY KEY (ID)
);
CREATE TABLE PERSON (
        LNAME                   VARCHAR (500)
        PERID                   VARCHAR (500) NOT NULL,
        FK_BOOKCATALOG          INT,
        ROOT_ELM                VARCHAR (500)
        ID                      INT,
        FOREIGN KEY (FK_BOOKCATALOG)REFERENCES BOOKCATALOG (ID),
        PRIMARY KEY (ID)
);
CREATE TABLE BOOK (
        FK_BOOKCATALOG          INT,
        BOOKTITLE               VARCHAR (500),
        FK_BOOKCATALOG          INT,
        ROOT_ELM                VARCHAR (500),
        ID                      INT,
        FOREIGN KEY (FK_BOOKCATALOG) REFERENCES BOOKCATALOG (ID),
        PRIMARY KEY (ID)
);

CREATE TABLE AUTHORS (
        AUTHORS                 VARCHAR (500) NOT NULL,
        FK_BOOK                 INT,
        ID                      INT,
        FOREIGN KEY (FK_BOOK) REFERENCES BOOK (ID)
        PRIMARY KEY (ID)
);
CREATE TABLE OTHER_NODES (
        FK_PERSON INT,
        ID                      INT,
        OTHER_VALUES            VARCHAR (500),
        ELEMENT_NAME            VARCHAR (500),
        FK_BOOKCATALOG          INT,
        ROOT_ELM                VARCHAR (500),
        FK_BOOK                 INT,
        FK_PUBLISHER            INT,
        PARENT_ELM              VARCHAR(500),
        FOREIGN KEY (FK_PERSON) REFERENCES PERSON (ID)
        FOREIGN KEY (FK_BOOKCATALOG) REFERENCES BOOK (ID)
        FOREIGN KEY (FK_BOOK) REFREENCES BOOK (ID),
        FOREIGN KEY (FK_PUBLISHER) REFREENCES PUBLISHER (ID),
        PRIMARY KEY (ID)
);
```

(그림 11) 결과 SQL Script

4. 실험 및 성능평가

본 장에서는 XML 문서의 공통구조 추출을 통한 관계형 데이터베이스 스키마 추출 기법의 성능평가를 위한 시스템 구현 환경과 실험 수행 결과를 설명한다. 성능평가를 위한 실험은 관련 연구에서 언급된 가장 일반적이면서 성능 면에서 우수한 방법인 Hybrid Inlining Algorithm[1]을 비교 대상으로 하였으며, 시스템 구현 과정에서 참조한 XMLDBMS[14] 방법을 함께 비교 대상으로 설정한다. 1절에서는 구현환경에 대하여 기술하고, 2절에서는 XMLDBMS와 Hybrid Inlining Algorithm과의 비교를 통해 성능평가 내용을 기술한다.

4.1 구현 환경

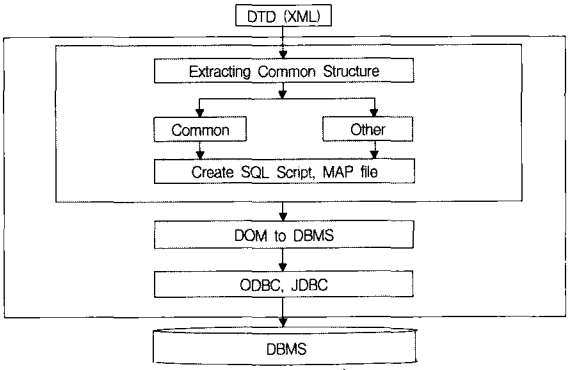
공통구조 추출을 위한 알고리즘의 구현은 관련 연구에서 언급한 XMLDBMS JAVA Packages의 소스 코드를 수정하여 개발하였다. 다음은 구체적인 구현환경이다.

- Window 2000 Server
- JDK 1.3.1_02
- Oracle XML Parser 2.0.2.9
- MS SQL Server 2000
- XMLDMS JAVA Packages

XMLDBMS는 XML 문서와 관계형 데이터베이스 시스템 사이에서 데이터 교환을 담당하는 미들웨어 역할을 하는 JAVA Packages 배포본으로 누구나 수정하여 재 배포가 가능하다.

XMLDBMS는 관련 연구에서 설명한 고유의 알고리즘을 토대로 XML 문서의 DTD를 관계형 데이터베이스 스키마로 변환하는 부분과 실제 XML 문서의 데이터를 관계형 데이터베이스 테이블로 맵핑하는 부분이 존재한다.

본 논문에서는 XMLDBMS의 소스를 수정하여 공통구조를 추출하는 모듈을 작성하였다. 또한, 실제 관계형 데이터베이스 스키마를 생성하는 부분을 위해 공통 릴레이션 스키마 추출 부분과 비 공통 릴레이션 스키마 추출 부분으로 나누어 모듈을 작성하였다.



(그림 12) 시스템 구조

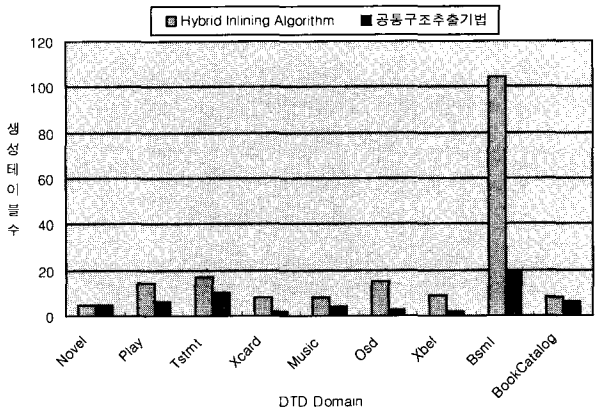
(그림 12)는 XMLDBMS를 수정하여 구성한 시스템 구성도이다. 본 시스템은 XML 문서의 DTD를 입력받아 유효성을 검증한 후 공통구조와 비 공통구조를 추출한다. 각각의 공통구조와 비 공통구조는 관계형 데이터베이스 스키마 추출 모듈을 통해 SQL Script를 생성한다. 생성된 SQL문을 이용하여 데이터베이스 시스템에 테이블을 생성한 후, DOMtoDBMS 모듈을 통해 XML 문서의 데이터들을 관계형 데이터베이스 시스템의 테이블로 맵핑한다.

4.2 성능평가

본 절에서는 기존의 방법들과의 성능비교를 위해서 XML 문서의 DTD를 관계형 데이터베이스 스키마로 변환했을 때 생성된 테이블 수를 비교한다. 또한, 전체 테이블 크기와 질의 처리시에 참조 테이블 수, 질의 처리시의 참조 테이블의 레코드 크기, 질의수행 시간으로 나누어서 성능비교를 수행한다.

4.2.1 생성 테이블 수 비교

본 논문에서는 테이블 생성 수를 비교하기 위해 [12]에서 CPI[15] 알고리즘을 구현하기 위해 사용했던 DTD들을 사용했다. 또한 CPI[15]는 기본적으로 Hybrid Inlining Algorithm[1]을 스키마 추출 기법으로 사용하고 있다.



(그림 13) DTD와 생성 테이블 수

<표 4>은 DTD의 자세한 구조를 엘리먼트와 애트리뷰트로 나누어 각각의 카디널리티 연산자의 수와 생성 테이블 수를 나타낸다. (그림 13)과 <표 4>에 기술된 DTD들은 <표 5>에 각각의 도메인에 대한 설명이 있다. (그림 13)은 XML 문서의 DTD를 관계형 데이터 베이스 스키마로 변환했을 때 생성되는 테이블 수를 나타낸다. 결과에서 알 수 있듯이 공통구조 추출기법이 가장 적은 수의 테이블을 생성한다.

Novel에서 Hybrid와 공통구조 추출기법이 동일한 테이블 수를 나타내는 이유는 Novel DTD에는 비 공통구조가 존재하지 않기 때문이다. 또한, XMDBMS에서 '-'로 나타난 부분은 XMLDBMS가 복잡한 DTD구조를 지원하지 않기 때문이다. 즉, 순환관계를 나타내는 구조가 DTD에 존재 할 경우 결과 관계형 스키마를 생성하지 못하였다.

<표 4> DTD와 생성 테이블 수

	DTD 구조										관계형 스키마		
Name	Ele 수	Att 수	Element 노드 구조				Attribute 노드 구조				생성 테이블 수		
			+	*	?	n	Att	ID	IDREF	IDREFS	XML DBMS	Hybrid	공통 구조
Novel	10	1	3	0	0	7	0	1	1	0	7	5	5
Play	21	0	10	1	3	7	0	0	0	0	-	14	6
Tstmt	28	0	12	1	8	7	0	0	0	0	-	17	10
Xcard	23	1	2	3	6	12	1	0	0	0	14	8	2
Music	12	17	2	5	0	5	17	0	0	0	11	8	4
Osd	16	15	0	13	0	3	15	0	0	0	16	15	3
Xbel	9	13	1	4	3	1	9	3	1	0	-	9	2
Bsml	112	2495	7	44	32	29	2314	84	81	16	-	104	19
Book Catalog	18	2	3	1	6	8	0	1	0	1	9	8	6

〈표 5〉 DTD Domain

DTD	Domain
Novel	Literature
Play	Shakespears Play
Tstmt	Religious Text
Xcard	Business Card
Music	Music Description
Osd	S/W Description
Xbel	Bookmark
Bsml	DNA Sequencing
BookCatalog	Book Catalog

실제로 테이블 수 적다는 것만으로는 성능이 우수하다고 단정할 수 없다. 동일한 데이터 양을 가지고 적은 수의 테이블이 생성된다는 것은 어느 한곳으로 데이터가 집중된다는 의미가 된다. 예를 들어, 공통구조 추출 기법에서 공통구조가 비 공통구조에 비해 크기가 작아진다면 실제 질의 처리 시에 크기가 큰 비 공통구조 테이블을 많이 검색 할 것이기 때문에 더 많은 비용을 필요로 할 것이다. 다음절에서는 이러한 사항을 논의하기 위해 질의 처리시 참조 테이블 수와 레코드 크기를 비교한다.

4.2.2 테이블 크기와 질의 처리시 참조 테이블 수 비교

본 절에서는 예제로 사용한 BookCatalog DTD를 사용하는 XML 문서들을 XMLDBMS와 Hybrid Inlining Algorithm, 공통구조 추출기법을 사용하여 실제 SQL Server 2000 데이터베이스 시스템에 테이블을 생성하여 XML 데이터를 입력 후, 생성 테이블의 전체 크기와 질의 수행시 참조 테이블 수, 질의 수행시 참조 테이블의 크기를 비교한다.

〈표 6〉 테이블 크기를 위한 레코드 크기

• BookCatalog 테이블 (1004byte) BOOKCATALOG_ CATALOGTITLE (500) ID (4) ROOT_ELM (500) • Person 테이블 (1508byte) LNAME (500) PERID (500) FK_BOOKCATALOG (4) ROOT_ELM (500) ID (4) • Publisher 테이블 (1008byte) FK_BOOKCATALOG (4) PUBLISHER_CORPNAME (500) ROOT_ELM (500) ID (4)	• Book 테이블 (1008byte) FK_BOOKCATALOG (4) BOOKTITLE (500) ROOT_ELM (500) ID (4) • Authors 테이블 (508byte) FK_BOOK (4) AUTHORS (500) ID (4) • Other_nodes 테이블 (580byte) FK_BOOKCATALOG (4) FK_PUBLISHER (4) FK_PERSON (4) FK_BOOK (4) ID (4) OTHER_VALUES (500) ELEMENT_NAME (20) ROOT_ELM (20) PARENT_ELM (20)

<표 6>의 테이블 길이 측정을 위한 각 레코드의 크기는 공통구조 추출기법의 테이블 구조를 한 예로 나타낸 것이다. XMDBMS와 Hybrid Inlining Algorithm에서도 실제 데이터를 나타내는 필드는 500byte이며 ID나 foreign key 필드는 4byte로 설정했다.

〈표 7〉 테이블 전체 레코드 크기

테이블 이름	테이블 크기		
	XMLDBMS	Hybrid	공통구조 추출기법
BookCatalog	1,536	3,012	3,012
Person	4,608	18,072	13,572
Publisher	6,120	9,048	6,048
Book	9,144	12,048	6,048
Authors	6,069	6,048	6,048
WebSite	2,032	4,032	-
Subject	1,524	3,024	-
Email	3,556	7,056	-
Name	9,144	-	-
Other_nodes	-	-	15,080
합 계	43,733	62,340	49,760

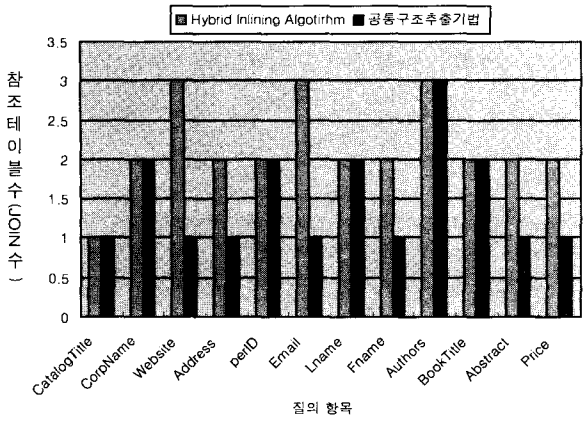
<표 7>은 XMLDBMS와 Hybrid Inlining Algorithm, 공통구조 추출기법 각각의 방법에 BookCatalog.dtd를 사용하는 3개의 XML 문서의 데이터를 입력 후의 전체 테이블의 크기를 나타낸다. 실제 테이블의 크기는 결과와 같이 XMLDBMS 방법이 가장 적은 크기를 가진다. 하지만 XMLDBMS는 순환관계를 지원하기 위한 ROOT_ELM 필드나 PARENT_ELM 필드를 포함하지 않기 때문에 적은 크기의 레코드들을 가진다. 그렇기 때문에, 공통구조 추출기법이나 Hybrid 방법보다는 보다 적은 테이블 크기를 가진다. 결과적으로는 공통구조 추출기법의 실제 레코드의 크기가 기존의 방법인 Hybrid 보다 적게 소요된다.

〈표 8〉 질의 종류와 처리에 따른 참조 테이블 수

질 의	참조 테이블 수		
	XMLDBMS	Hybrid	공통구조 추출기법
BookCatalog/CatalogTitle	1	1	1
BookCatalog/Publisher/CorpName	2	2	2
BookCatalog/Publisher/WebSite	3	3	1
BookCatalog/Publisher/Address	2	2	1
BookCatalog/Person/perID	2	2	2
BookCatalog/Person/Email	3	3	1
BookCatalog/Person/Name/LName	3	2	2
BookCatalog/Person/Name/FNname	3	2	1
BookCatalog/Book/Authors	3	3	3
BookCatalog/Book/BookTitle	2	2	2
BookCatalog/Book/Abstract	2	2	1
BookCatalog/Book/Price	2	2	1

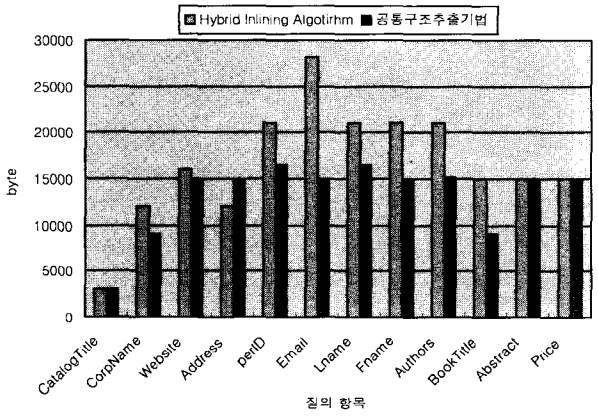
<표 8>은 질의에 따른 참조 테이블 수의 비교를 위하여 사용한 질의의 종류와 각각의 질의 처리에 필요한 참조 테이블의 수를 나타낸 것이다. 예를 들어 XMLDBMS와 Hybrid에서 특정 아이디를 갖는 Address를 검색 하고자 할 때 다수의 BookCatalog와 Publisher중 찾고자하는 Address와 일치하는 레코드를 찾아야 하기 때문에 참조 테이블(조인 테이블)은 BookCatalog와 Publisher 두 개가 된다. 하지만, 공통구조 추출기법에서는 Address가 비 공통구조 스키마에 해당하기 때문에 하나의 테이블만을 참조한다. 비 공통 릴

레이션 스키마는 부모 테이블 노드에 대한 정보를 외래키로 연결하기 때문에 하나의 테이블만으로 데이터 검색이 가능하다. (그림 14)는 Hybrid Inlining Algorithm과 공통구조 추출 기법의 참조 테이블 수를 비교한 그래프이다.



(그림 14) 질의에 따른 참조 테이블 수

그러나, 비 공통 테이블의 레코드 크기가 비대하다면 실제 데이터 검색 시간은 XMLDBMS나 Hybrid Inlining Algorithm이 성능 면에서 우수 할 수 있다. 이를 비교하기 위하여 (그림 15)는 질의 처리시에 참조 테이블의 레코드 크기를 측정한 결과이다. 실제 테이블 크기는 앞서 언급한 것처럼 XMLDBMS가 가장 적지만 XMLDBMS는 실제로 복잡한 DTD를 지원 할 수 없기 때문에 테이블 레코드 크기가 기본적으로 적기 때문에 비교 대상이 아니다. 결과를 살펴보면 공통구조 추출기법이 Hybrid Inlining Algorithm 보다 거의 모든 질의에서 적은 테이블 레코드 크기를 갖는다는 것을 알 수 있다.

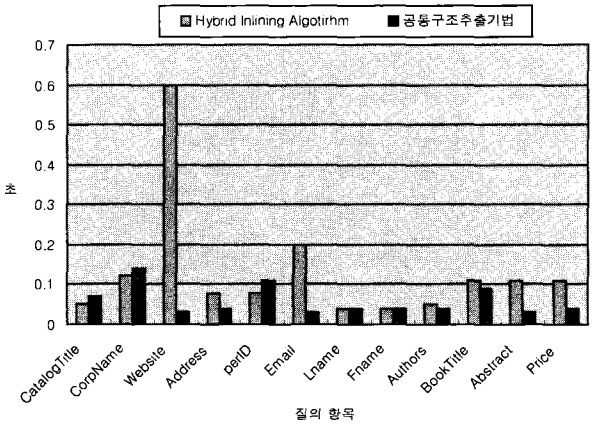


(그림 15) 질의에 따른 참조 테이블 레코드 크기

4.2.3 질의 처리 시간 비교

본 절에서는 Hybrid Inlining Algorithm과 공통구조 추출 기법 각각의 방법에 BookCatalog.dtd를 사용하는 3개의 XML 문서의 데이터를 입력 후 앞 절에서 사용했던 질의에 대한 응답 시간을 측정하였다.

각각의 질의에 대한 응답 시간은 각각의 질의를 백 회 수행한 결과 시간이다. 각각의 테이블의 인덱스 구성시 결과인 (그림 16)에서 CatalogTitle, CorpName, perID 노드들을 제외하고 다른 모두 노드들은 공통구조 추출기법이 Hybrid보다 질의 처리 시간이 짧은 결과를 보였다. 또한, 비 공통 스키마에 포함되는 노드인 Website, Address, Email, FName, Abstract, Price는 Hybrid 방법에 비해 질의 처리 시간이 현저히 감소하는 결과를 확인 할 수 있다. 결과적으로 생성 테이블 수와 질의 처리 결과 시간을 포함한 모든 성능 평가 부분에서 공통구조 추출 기법이 기존의 알고리즘 보다 좋은 결과를 얻을 수 있다



(그림 16) 질의 처리 시간 비교

5. 결론 및 향후 연구 과제

XML은 최근 웹 상에서 데이터 교환의 표준 형식으로 주목을 받고 있다. XML은 사용자가 문서의 구조를 정의 할 수 있기 때문에 다양한 형태의 데이터를 XML 문서로 표현할 수가 있다. 이러한 점은 XML이 이 기종의 컴퓨터사이에서 데이터 교환의 매체로 사용 될 수 있다는 것을 암시한다.

XML문서를 질의 처리하기 위한 방법으로 XML 문서 전용 질의 언어가 개발되고 있지만, 데이터의 양이 증가한다면 결국 막대한 양의 데이터를 처리 할 데이터베이스 시스템을 필요로 하게 된다. XML 데이터를 저장하고 질의하기 위한 데이터베이스 시스템은 XML 전용 데이터 베이스 시스템과 기존의 관계형 데이터베이스 시스템으로 나누어 볼 수 있다. XML 전용 데이터베이스 시스템은 저장 장치에서 질의 처리기까지 모두 개발해야 한다는 점과 개발 후 시스템의 안정성이나 효율성에 대한 평가가 필요하다.

본 논문에서는 기존의 관계형 데이터베이스 시스템의 장점인 질의처리 기법과 넓은 시장성을 고려하여, XML 데이터를 관계형 데이터베이스 시스템에 저장하는 기법을 제안하였다. 제안 기법인 공통구조 추출 기법은 XML 문서의 DTD를 이용하여 XML 문서의 공통구조를 추출하여 공통구조와 비 공통구조를 각각 관계형 데이터베이스 스키마로 변환하는 기법이다. 공통구조는 동일한 DTD를 사용하는

모든 XML 문서에 반드시 존재하는 노드들로 구성되며, 비공통구조는 동일한 DTD를 사용하는 XML 문서들에 불규칙적으로 등장하는 노드들이다.

본 논문에서는 제안된 공통구조 추출 기법을 실제로 구현하여 기존의 방법들과 성능의 비교 분석을 수행하였다. 성능분석 결과에서 제안된 기법은 생성 테이블의 수를 줄이고 널(NULL)값의 출현을 감소시켜, 결과적으로 질의 처리 시 우수한 성능을 나타낼 수 있음을 확인하였다.

XML 문서를 관계형 데이터 베이스 시스템으로 맵핑하는 방법에 있어서는 DTD의 구조가 가장 중요한 역할을 한다. 기존의 방법은 DTD에 '*', '?' 연산자가 많이 존재한다면, 하나의 XML 문서가 다수의 테이블로 분산되고 각각의 테이블은 많은 수의 NULL 값을 가지게 되며 이는 성능저하와 비용 증대로 이어진다. 본 논문에서 제안한 공통구조 추출기법은 이러한 점을 극복하기 위한 것이다.

하지만 DTD 설계자가 상위 노드에 '*'와 '?' 연산자를 주로 사용하여 설계한다면, 공통구조가 전체 데이터에 비해 작아질 것이다. 이러한 점은 대부분의 데이터가 하나의 테이블에 집중되는 결과를 낳는데 이는 성능저하로 이어진다. 공통구조 추출 기법의 단점으로 나타나는 데이터 집중화 현상을 극복하기 위한 방법으로는 '*'와 '?' 연산자가 상위 노드에 임의의 임계값보다 많은 수가 발생한다면 연산자 변환 시에 트리 구조의 Level을 고려하는 방법을 고려할 수 있다. 본 논문의 향후 연구 방안은 공통구조 추출기법의 단점인 데이터 집중화 현상을 극복할 수 있는 방법 모색에 집중 될 것이며 DTD의 단점을 보완하기 위해 XML 문서의 스키마로 새롭게 등장한 XML Schema를 지원하는 방안 또한 향후 연구의 주요 초점이 될 것이다.

참 고 문 헌

[1] J. Shanmugasundaram, H. Gang, K. Tufte, C. Zhang, D. J. DeWitt, and J. F. Naughton, "Relational Databases for Querying XML Documents : Limitation and Opportunities," Proc. of VLDB, Edinburgh, Scotland, pp.302-304, 1999.

[2] T. Shimura, M. Yoshikawa, and S. Uemura, "Storage and Retrieval of XML Documents Using Object-Relational Databases," DEXA, 1999.

[3] A. Deutsch, M. F. Fernandez, and D. Suciu, "Storing Semi-structured Data with STORED," Proc. of ACM SIGMOD Conference, 1999.

[4] D. Florescu and D. Kossmann, "Storing and Querying XML Data Using an RDBMS," Proc. of Int. Conf. on Data Eng., 1999.

[5] C. Kanne and G. Moerkotte, "Efficient Storage of XML Data," Proc. of Int. Conf. on Data Eng., 1998.

[6] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom, "Lore : A Database Management System for Semi Structured Data," Technical Report, Stanford University Database Group, February, 1997.

[7] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Wisdom, "The TSI-MMIS Project : Integration of Heterogeneous Information Sources," Proc. of IPSJ Conference, pp.7-18, 1994.

[8] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu, "A Query Language for XML," Proc. of 8th International World Wide Web Conference, 1999.

[9] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu, "A Query Language and Optimization Technique for Unstructured Data," Proc. of ACM SIGMOD International Conference on Management of Data, 1996.

[10] J. McHugh and J. Widom, "Query Optimization for XML," Proc. of Very Large Data Bases, Edinburgh, U.K., 1999.

[11] M. N. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim, "XTRACT : A System for Extracting Document Type Descriptors from XML Documents," Proc. of ACM SIGMOD Conference on Management of Data, Dallas, Texas, May, 2000.

[12] D. W. Lee and W. W. Chu, "Constraints-preserving Transformation from XML Documents Type Definition to Relation Schema," UCLA-CS-TR, 2000.

[13] A. Bonifati and S. Ceri, "Comparative Analysis of Five XML Query Languages," ACM SIGMOD Record, 29(1), 2000.

[14] http://www.rpbourret.com/xmldbms/index.htm/.

[15] http://www.cobase.cs.ucla.edu/projects/xpress/.

[16] T. Bray, J. Paoli, and C. M. Sperberg-McQueen, "Extensible Markup Language (XML) 1.0 (Second Edition)," http://www.w3.org/TR/REC-xml, W3C Recommendation 6, October, 2000.

안 성 은

e-mail : bomber@mail.kangwon.ac.kr

2000년 강원대학교 기계공학과 졸업(학사)

2002년 강원대학교 컴퓨터정보통신공학과 (석사)

관심분야 : 데이터베이스 시스템, XML 등

최 황 규

e-mail : hkchoi@kangwon.ac.kr

1984년 경북대학교 전자공학과(학사)

1986년 한국과학기술원 전기및전자공학과 (석사)

1989년 한국과학기술원 전기및전자공학과 (박사)

1994년~1995년 Univ. of Florida Database R&D Center 방문교수

1999년~2001년 강원대학교 전자계산소 소장

1990년~현재 강원대학교 전기전자정보통신공학부 교수

2002년~ 현재 Univ. of Minnesota 방문교수

관심분야 : 멀티미디어 시스템, 데이터베이스 시스템, XML 응용, Intelligent Storage System 등